Computation in an algebra of test selection criteria

Jan Pachl* and Shmuel Zaks†

May 12, 1993

Abstract

One of the key concepts in testing is that of adequate test sets. A *test selection criterion* decides which test sets are adequate. In this paper, a language schema for specifying a large class of test selection criteria is developed; the schema is based on two operations for building complex criteria from simple ones. Basic algebraic properties of the two operations are derived.

In the second part of the paper, a simple language — an instance of the general schema — is studied in detail, with the goal of generating small adequate test sets automatically. It is shown that one version of the problem is intractable, while another is solvable by an efficient algorithm. An implementation of the algorithm is described.

Note added on December 23, 1999 {Jan Pachl}: This version is dated May 12, 1993. A previous version was issued as an IBM research report RZ 2114, dated April 16, 1991. The paper has not been published.

*IBM Research Division, Zurich Research Laboratory, Säumerstrasse 4, 8803 Rüschlikon, Switzerland

†Department of Computer Science, Technion, Haifa, Israel

Contents

1 Introduction

This paper deals with testing of computer programs. However, most of our discussion applies to testing of more general systems.

Testing consists of experiments, called *tests*, in which the behavior of the system under test is compared to its **specification**. The system is often called an **implementation under test**; the purpose of testing is to conclude whether the system implements the specification.

The test designer must decide, possibly with machine assistance, what tests are to be executed and in what order. In this paper we assume that tests are repeatable and that the behavior of the implementation under test in each individual test does not depend on the order in which the tests are executed. Therefore the test designer's decision is described by a *set of tests*, selected from some set of tests that *could* be executed. To model this situation, we denote by D the **test domain**, i.e. some given set of tests for the implementation under test. Subsets of D are called **test sets**.

An important concept is that of adequate test sets. Informally, a subset T of D is **adequate** if we believe that it is sufficient to execute the tests in T, instead of all the tests in D. Once we have checked that the behavior of the implementation satisfies the specification for each test d in T, we are willing to accept that the same will be true for each d in D. To make this concept independent of subjective beliefs, we define adequacy with respect to a test selection criterion: A **test selection criterion on** D is a rule that decides for each subset T of D whether T is adequate or not. (Other terms have been used in the literature, e.g. *data selection criterion* [5], *test method* [6], *testing method* [7]). A test selection criterion may be defined based on the knowledge of the implementation under test, of its specification, or both; Gourlay [6] introduced a framework for discussing these dependencies explicitly.

Many natural test selection criteria can be described as follows: There is a collection of subsets of the domain D, and $T \subseteq D$ is adequate if and only if T intersects every nonempty set in the collection. The following three examples of selection criteria from the literature, and many others, are of this form.

1. **Condition table method** [5]. "[I]dentify conditions describing some aspect of the

problem or program to be tested" ([5], p. 167), and then combine the conditions to form test predicates on D, the set of inputs. A test set T is *complete* ([5], p. 170) if

- for each thus formed test predicate there is a point in T that satisfies the predicate; and
- each point in T satisfies at least one of the predicates.

The first condition is clearly the adequacy of T as described above, with respect to a collection of subsets of D.

2. **Cause-effect graphing** [3, 10]. A *cause-effect graph* is a simplified specification of the system under test. Nodes in the graph represent important properties of *causes* (inputs) and *effects* (outputs) and possibly additional intermediate properties. Edges represent how the effects depend on the causes. Once the cause-effect graph has been constructed, it can be used for systematic selection of a set of inputs for testing. Let N be the set of nodes in the graph. Each input defines a subset of N; thus the domain D corresponds to a set of subsets of N. One simple test selection criterion is:

   - Ensure that each effect node is covered at least once.

   This is clearly adequacy as described above, with respect to a collection of subsets of D. Myers ([10], pp. 65-68) described a more complex test selection criterion based on the cause-effect graph; again his description can be defined as adequacy with respect to a collection of subsets of D.

3. **Statement coverage** [10]. Let the implementation under test be implemented by a program consisting of a number of statements. For each statement s in the program, let X_s be the set of the tests in D that cause s to be executed. Then $T \subseteq D$ is adequate with respect to the collection $\{X_s\}$ if and only if T covers every statement covered by D.

Jeng and Weyuker [9] give several other examples of test selection criteria of this general form, which they call *partition testing*.

In the present paper we describe a simple but powerful language for specifying test selection criteria; the language is based on our previous proposal [11]. A language for

specifying test selection criteria is needed when we wish to free the test designer from dealing with individual test cases. The test designer should be able to specify what constitutes an adequate test set in a high-level notation, from which individual test cases are then generated automatically.

Balcer, Hasling and Ostrand [2] built a system called TSL, which supports this high-level approach to testing. Our design can serve as a model for extending the test specification language in TSL, and for defining other similar languages. We return to the comparison with TSL in Section 7.2.

We describe a general language *schema*, from which concrete languages are derived by choosing types of parameters. The schema is based on two operations for combining selection criteria; with these two operations, test selection criteria form a well-behaved algebra. The ability to combine criteria using the two operations yields a number of benefits:

- The language has simple well-defined semantics.
- The language is powerful — many useful criteria can be expressed in the language.
- Algorithms that process criteria and generate test sets can use algebraic identities to manipulate criteria.

In the second half of the paper we define one language based on the general schema, and study the algorithms that generate adequate test sets for the criteria expressed in the language. We show that the problem of finding a minimum adequate test set (i.e. an adequate test set of the smallest size) is NP-hard, and then we concentrate on the problem of finding a minimal adequate test set (i.e. a test set whose proper subsets are not adequate). We also describe what we learned from implementing a prototype tool for generating minimal adequate test sets.

Related work and topics for further research are discussed in the last section.

2 Example

To illustrate the concept of a test selection criterion, we now describe a simple testing scenario, adopted from the paper by Balcer, Hasling and Ostrand [2].

declaration
separator_1 : { "/", "z" }
separator_2 : { "/", "x" }
string_1 : { "", "a", "ab", "abcd", "abcd987", "abcdefghijklmnopqrstuvwxyz0123" }
string_2 : { "", "a", "ab", "abcd", "abcd987", "abcdefghijklmnopqrstuvwxyz0123" }
string_1_occurs : { true, false }

Figure 1: Parameter declarations for the example

Test suites typically consist of many test cases that differ only slightly from each other. Rather than preparing all the variations one by one, the test designer may prepare a "parameterized test case" (a "code template" in the terminology of [2]) and then generate individual test cases by systematically filling in the values of the parameters.

In the sample scenario, a text editor is to be tested against the specification of the CHANGE command. The syntax of the command is

```
C /string1/string2
```

As in [2], the parameterized test case for this task uses five parameters. (More precisely, the TSL description in [2] uses four parameters and one environment condition; however, the distinction is not important for our discussion.)

Parameter declarations are in Figure 1. To obtain one individual test case, we select one value for each parameter, and substitute the selected values to

`C` **separator_1 string_1 separator_2 string_2**

The value of the parameter **string_1_occurs** is used to set up the current line in the editor (so that it does or does not contain **string_1**).

Now observe that the parameter declarations in Figure 1 define a test domain D: Each combination of values for the five parameters defines a test in D. In some cases it may be feasible to execute all tests in D. However, even in our simple example D contains $2 \times 2 \times 6 \times 6 \times 2 = 288$ elements. It is easy to imagine much larger examples, for which testing with all inputs in D would be infeasible. The test designer must then select a **test set**, i.e. a subset of D. Sometimes the test designer wants to list the points of the test

set explicitly, one by one. However, it is frequently more convenient to write a high-level description of a test selection criterion, and let an automated tool select a test set adequate for the criterion.

Let us consider several examples of high-level descriptions of test selection criteria that free the test designer from the need to think in terms of individual test cases. For our domain D, the criterion

$$\langle\ \textbf{string\_1} = \text{"}\texttt{a}\text{"}\ \rangle \tag{1}$$

specifies that the test set must include at least one point in which the value of the parameter **string_1** is "a". The criterion

$$\mathsf{EACH}(\ \textbf{string\_1}\ :\ \text{"}\texttt{a}\text{"},\ \text{"}\texttt{ab}\text{"},\ \text{"}\texttt{abcd987}\text{"}\ ) \tag{2}$$

specifies that for each of the three listed values of the parameter **string_1** the test set must include at least one point with that value. It is convenient to have another primitive as an abbreviation for EACH whose arguments include all values declared for the parameter; the primitive EXHAUSTIVE with one argument has this role. Thus

$$\mathsf{EXHAUSTIVE}(\ \textbf{string\_1}\ ) \tag{3}$$

has the same meaning as

$$\mathsf{EACH}(\ \textbf{string\_1}\ :\ \text{""},\ \text{"}\texttt{a}\text{"},\ \text{"}\texttt{ab}\text{"},\ \text{"}\texttt{abcd}\text{"},\ \text{"}\texttt{abcd987}\text{"},\ \text{"}\texttt{abcdefghijklmnopqrstuvwxyz0123}\text{"}\ )\ . \tag{4}$$

As we shall see in the next section, (3) and the criterion

$$\mathsf{EXHAUSTIVE}(\ \textbf{separator\_1}\ ) \tag{5}$$

can be combined in two basic ways. One combination is

$$\mathsf{EXHAUSTIVE}(\ \textbf{string\_1}\ )\ \otimes\ \mathsf{EXHAUSTIVE}(\ \textbf{separator\_1}\ )\ ,$$

which can be also written as

$$\mathsf{EXHAUSTIVE}(\ \textbf{string\_1},\ \textbf{separator\_1}\ )\ .$$

It specifies that all possible combinations of the values of **string_1** and **separator_1** must be included; since **string_1** assumes six values and **separator_1** two values, any test set adequate for this criterion must contain at least 12 elements. The other combination of (3) and (5) is

$$\mathsf{EXHAUSTIVE}(\ \textbf{string\_1}\ )\ \uplus\ \mathsf{EXHAUSTIVE}(\ \textbf{separator\_1}\ )\ ,$$

which merely requires that the test set must be adequate for (3) and also for (5). A test set containing 6 points is sufficient for that; for example, the following six combinations of **string_1** and **separator_1** are sufficient:

	string_1	**separator_1**
1.	""	"/"
2.	"a"	"/"
3.	"ab"	"/"
4.	"abcd"	"/"
5.	"abcd987"	"/"
6.	"abcdefghijklmnopqrstuvwxyz0123"	"z"

In the next section we describe a more systematic approach to the construction of test selection criteria. We shall see that many complex criteria, including EACH and EXHAUSTIVE, may be constructed from simple ones.

3 A language for test selection criteria

3.1 A general language schema

We are now going to describe a language for specifying instances of the test selection problem. We start by describing a general **language schema**. Many different concrete languages may then be obtained from the schema by allowing different parameter types. One such choice of parameter types and the resulting concrete language are discussed in Section 3.3 and in the rest of the paper.

To define an instance of the test selection problem, we have to specify a domain D and a test selection criterion on D. In our approach, D and the criterion on D have the following special form:

- D is a subset of the Cartesian product $P = \prod_{i=1}^{N} Q_i$ of certain sets $Q_1, \ldots, Q_N$. The points in P are vectors $(v_1, \ldots, v_N)$ of parameter values $v_i \in Q_i$.

- The criterion is defined by a set of subsets of P.

Thus to define an instance of the test selection problem, we specify sets Q_i , a subset D of the product P of Q_i , and a set of subsets of P. In our language, the specification consists of three parts:

1. declaration of parameters;

2. a constraint;

3. a test selection criterion.

Part 1 defines the sets Q_i , part 2 the set D, and part 3 the set of subsets of P.

The first part, denoted Δ, is a set of **declarations**

$$q_i \;:\; Q_i$$

each of which declares a **parameter** q_i and its **range** Q_i. Define

$$P(\Delta) = \prod_{i=1}^{N} Q_i \quad .$$

For example, for the declarations in Figure 1, $P(\Delta)$ is the Cartesian product of five sets Q_i:

$$\begin{aligned}
Q_1 &= \{\text{"}\texttt{/}\text{"}, \text{"}\texttt{z}\text{"}\} \\
Q_2 &= \{\text{"}\texttt{/}\text{"}, \text{"}\texttt{x}\text{"}\} \\
Q_3 = Q_4 &= \{\text{""}, \text{"}\texttt{a}\text{"}, \text{"}\texttt{ab}\text{"}, \text{"}\texttt{abcd}\text{"}, \text{"}\texttt{abcd987}\text{"}, \text{"}\texttt{abcdefghijklmnopqrstuvwxyz0123}\text{"}\} \\
Q_5 &= \{\mathsf{true}, \mathsf{false}\}
\end{aligned}$$

The second part is a **constraint**; it is a boolean expression $\psi = \psi(q_1, \ldots, q_N)$ built from **primitive constraints** by means of binary operators $\vee$ (logical or) and $\wedge$ (logical and). To interpret the constraint, we have to assign the value true or false to each primitive constraint

in the expression when arbitrary values $(v_1, \ldots, v_N)$ are substituted for the parameters $(q_1, \ldots, q_N)$. The constraint then defines the domain

$$D(\Delta, \psi) = \{\ (v_1, \ldots, v_N) \in P(\Delta)\ \mid\ \psi(v_1, \ldots, v_N) = \mathsf{true}\ \}\ .$$

We write $D(\psi)$ instead of $D(\Delta, \psi)$ when no misunderstanding is possible.

The example in Section 2 does not specify any constraint, and therefore $D(\psi) = P(\Delta)$.

The third part is a **test selection criterion**; it is an expression built from **primitive criteria** by means of binary operators $\uplus$ and $\otimes$. The value of such an expression Γ is a set $\mathsf{S}(\Delta, \Gamma)$ of subsets of $P(\Delta)$. Again we write $\mathsf{S}(\Gamma)$ instead of $\mathsf{S}(\Delta, \Gamma)$ when no misunderstanding is possible. Once the value $\mathsf{S}(\Gamma)$ has been defined for every primitive criterion Γ, we define $\mathsf{S}(\Gamma)$ for general Γ as follows: Given two criteria Γ_1 and Γ_2, define

$$\begin{aligned}
\mathsf{S}(\Gamma_1 \uplus \Gamma_2) &= \{\ X\ \mid\ X \in \mathsf{S}(\Gamma_1) \quad \text{or} \quad X \in \mathsf{S}(\Gamma_2)\ \} = \mathsf{S}(\Gamma_1) \cup \mathsf{S}(\Gamma_2)\ , \\
\mathsf{S}(\Gamma_1 \otimes \Gamma_2) &= \{\ X_1 \cap X_2\ \mid\ X_1 \in \mathsf{S}(\Gamma_1), X_2 \in \mathsf{S}(\Gamma_2)\ \}\ .
\end{aligned}$$

In our example in Section 2, when Γ is the primitive criterion

$$\langle\ \textbf{string\_1} = \text{"}\mathtt{a}\text{"}\ \rangle$$

the set $\mathsf{S}(\Gamma)$ contains a single subset of $P(\Delta)$, namely

$$\{\ (sep_1, sep_2, s_1, s_2, o) \in P(\Delta) \mid s_1 = \text{"}\mathtt{a}\text{"}\ \}\ .$$

Similarly, we could take EACH and $\mathsf{EXHAUSTIVE}$ as primitive criteria and define their values $\mathsf{S}(\Gamma)$; however, we shall see later that these criteria can be derived from simpler ones using $\uplus$ and $\otimes$.

Definition. An **instance of the test selection problem** is $I = (\ \Delta,\ \psi,\ \Gamma\ )$, where Δ is a set of parameter declarations, ψ is a constraint, and Γ is a test selection criterion. A set $T \subseteq D(\Delta, \psi)$ is **adequate for** I if $T \cap X \neq \emptyset$ for every $X \in \mathsf{S}(\Delta, \Gamma)$ such that $X \cap D(\Delta, \psi) \neq \emptyset$. We also say that T is **adequate for** Γ if Δ and ψ are understood from the context.

From the definition of $\Gamma_1 \uplus \Gamma_2$ it follows that a test set T is adequate for $\Gamma_1 \uplus \Gamma_2$ if and only if it is adequate for Γ_1 and also for Γ_2. The criterion $\Gamma_1 \uplus \Gamma_2$ is used when the test designer wants to satisfy Γ_1 and Γ_2 independently.

The criterion $\Gamma_1 \otimes \Gamma_2$ is used when the test designer suspects dependencies between Γ_1 and Γ_2, and wants to test for the faults produced by combinations of causes. If Γ_1 enforces the selection of a test point that has some property p_1 and Γ_2 the selection of a test point that has some property p_2, then the criterion $\Gamma_1 \otimes \Gamma_2$ enforces the selection of a test point with the property p_1 **and** p_2 (if such a point exists in $D(\psi)$).

Since $\mathsf{S}(\Gamma)$ is the value of the expression Γ, it is natural to write $\Gamma_1 = \Gamma_2$ when $\mathsf{S}(\Gamma_1) = \mathsf{S}(\Gamma_2)$, and $\Gamma_1 \subseteq \Gamma_2$ when $\mathsf{S}(\Gamma_1) \subseteq \mathsf{S}(\Gamma_2)$. It is a simple exercise to show that both $\uplus$ and $\otimes$ are commutative and associative, and that the following distributive law holds:

$$(\Gamma_1 \uplus \Gamma_2) \otimes \Gamma_3 = (\Gamma_1 \otimes \Gamma_3) \uplus (\Gamma_2 \otimes \Gamma_3) \ .$$

Since $\uplus$ and $\otimes$ are associative, we write expressions like $\Gamma_1 \uplus \Gamma_2 \uplus \Gamma_3$ and $\Gamma_1 \otimes \Gamma_2 \otimes \Gamma_3$ without parentheses. We also use the notation $\biguplus_{j=1}^{m} \Gamma_j$ for $\Gamma_1 \uplus \Gamma_2 \uplus \ldots \uplus \Gamma_m$, and similarly for $\bigotimes$.

3.2 Comparing criteria

In this section we define several relations for comparing test selection criteria. The definitions of this section are not used in the rest of the paper, but the concepts will illustrate some important properties of the algebra of test selection criteria.

The following relation $\sqsubseteq$ describes the notion that one criterion is less stringent than another.

Definition. Let S_1 and S_2 be two sets of subsets of a set P. Write $\mathsf{S}_1 \sqsubseteq \mathsf{S}_2$ if the following is true for every $T \subseteq P$: if $T \cap X \neq \emptyset$ for every nonempty $X \in \mathsf{S}_2$ then $T \cap X \neq \emptyset$ for every nonempty $X \in \mathsf{S}_1$. Write $\mathsf{S}_1 \simeq \mathsf{S}_2$ if $\mathsf{S}_1 \sqsubseteq \mathsf{S}_2$ and $\mathsf{S}_2 \sqsubseteq \mathsf{S}_1$. For a fixed Δ and criteria Γ_1 and Γ_2, write $\Gamma_1 \sqsubseteq \Gamma_2$ if $\mathsf{S}(\Delta, \Gamma_1) \sqsubseteq \mathsf{S}(\Delta, \Gamma_2)$, and $\Gamma_1 \simeq \Gamma_2$ if $\mathsf{S}(\Delta, \Gamma_1) \simeq \mathsf{S}(\Delta, \Gamma_2)$.

The proof of the following proposition follows directly from definitions. In view of part 1, $\Gamma_1 \sqsubseteq \Gamma_2$ if and only if $(\Delta, \mathsf{true}, \Gamma_2)$ subsumes $(\Delta, \mathsf{true}, \Gamma_1)$ in the terminology of Hamlet [7].

Proposition 3.1 *Let Δ be a fixed set of declarations. If Γ_1 and Γ_2 are two criteria then*

1. $\Gamma_1 \sqsubseteq \Gamma_2$ *if and only if every* $T \subseteq P(\Delta)$ *adequate for* $(\Delta, \mathsf{true}, \Gamma_2)$ *is also adequate for* $(\Delta, \mathsf{true}, \Gamma_1)$*;*

2. $\Gamma_1 \simeq \Gamma_2$ *if and only if* $(\Delta, \mathsf{true}, \Gamma_1)$ *and* $(\Delta, \mathsf{true}, \Gamma_2)$ *have the same adequate sets;*

3. $\Gamma_1 \subseteq \Gamma_2$ *implies* $\Gamma_1 \sqsubseteq \Gamma_2$. □

By part 3, $\Gamma_1 = \Gamma_2$ implies $\Gamma_1 \simeq \Gamma_2$. Although $\Gamma_1 \simeq \Gamma_2$ does not imply $\Gamma_1 = \Gamma_2$, Proposition 3.2 below shows that $\simeq$ and $=$ are closely related.

Let S be a set of subsets of a set P. A set $X \in \mathsf{S}$ is **minimal in** S if $X \neq \emptyset$ and

$$Y \in \mathsf{S},\ Y \subseteq X \ \text{ implies } \ Y = X \ \text{ or } \ Y = \emptyset \ .$$

Let $\mathsf{MIN}(\mathsf{S})$ be the set of all minimal $X \in \mathsf{S}$.

Proposition 3.2 *If* S, S_1 *and* S_2 *are finite sets of subsets of* P *then*

1. $\mathsf{MIN}(\mathsf{S}) \simeq \mathsf{S}$ *;*

2. $\mathsf{S}_1 \simeq \mathsf{S}_2$ *if and only if* $\mathsf{MIN}(\mathsf{S}_1) = \mathsf{MIN}(\mathsf{S}_2)$ *.*

Proof. 1. Since $\mathsf{MIN}(\mathsf{S}) \subseteq \mathsf{S}$, it follows that $\mathsf{MIN}(\mathsf{S}) \sqsubseteq \mathsf{S}$. Since S is finite, for every nonempty $Y \in \mathsf{S}$ there exists a minimal $X \in \mathsf{S}$ such that $X \subseteq Y$; therefore $\mathsf{S} \sqsubseteq \mathsf{MIN}(\mathsf{S})$ by the definition of $\sqsubseteq$.

2. If $\mathsf{MIN}(\mathsf{S}_1) = \mathsf{MIN}(\mathsf{S}_2)$ then by part 1 we get

$$\mathsf{S}_1 \simeq \mathsf{MIN}(\mathsf{S}_1) = \mathsf{MIN}(\mathsf{S}_2) \simeq \mathsf{S}_2 \ .$$

Assume $\mathsf{S}_1 \simeq \mathsf{S}_2$ and $X_1 \in \mathsf{MIN}(\mathsf{S}_1)$. We have $\mathsf{MIN}(\mathsf{S}_1) \simeq \mathsf{MIN}(\mathsf{S}_2)$ by part 1. Set $T_1 = P \backslash X_1$; thus $T_1 \cap X_1 = \emptyset$. By the definition of $\mathsf{MIN}(\mathsf{S}_1) \sqsubseteq \mathsf{MIN}(\mathsf{S}_2)$ there exists $X_2 \in \mathsf{MIN}(\mathsf{S}_2)$ such that $T_1 \cap X_2 = \emptyset$. Thus $X_2 \subseteq X_1$. Now by the same argument applied to $T_2 = P \setminus X_2$ there exists $X_1' \in \mathsf{MIN}(\mathsf{S}_1)$ such that $X_1' \subseteq X_2 \subseteq X_1$. Since X_1 is minimal in $\mathsf{MIN}(\mathsf{S}_1)$, we have $X_1 = X_1'$, and therefore $X_1 = X_2$. We have proved that every $X_1 \in \mathsf{MIN}(\mathsf{S}_1)$ belongs to $\mathsf{MIN}(\mathsf{S}_2)$. By symmetry we get $\mathsf{MIN}(\mathsf{S}_1) = \mathsf{MIN}(\mathsf{S}_2)$. □

It is easy to verify that

$$\begin{aligned} \Gamma_1 \sqsubseteq \Gamma_1' &\quad \text{implies} \quad \Gamma_1 \uplus \Gamma_2 \sqsubseteq \Gamma_1' \uplus \Gamma_2 \\ \Gamma_1 \simeq \Gamma_1' &\quad \text{implies} \quad \Gamma_1 \uplus \Gamma_2 \simeq \Gamma_1' \uplus \Gamma_2 \end{aligned}$$

However, $\Gamma_1 \simeq \Gamma_1'$ does *not* imply $\Gamma_1 \otimes \Gamma_2 \simeq \Gamma_1' \otimes \Gamma_2$. Thus, even for $\psi = \mathsf{true}$, to determine which test sets are adequate with respect to $(\Delta, \psi, \Gamma_1 \otimes \Gamma_2)$, it is not enough to know which test sets are adequate with respect to (Δ, ψ, Γ_1) and which are adequate with respect to (Δ, ψ, Γ_2).

3.3 Enumerated types

From the general language schema described in Section 3.1 we obtain a concrete language by specifying allowed parameter types. To specify a parameter type, we must describe

- the range;
- primitive constraints;
- primitive criteria.

In addition, we must supply rules to evaluate primitive constraints and primitive criteria, so that $D(\psi)$ and $\mathsf{S}(\Gamma)$ are defined for any ψ and Γ.

We use the following convention: If $\varphi = \varphi(q_1, \ldots, q_N)$ is a Boolean expression then $\langle\varphi\rangle$ is the criterion for which the value $\mathsf{S}(\langle\varphi\rangle)$ contains a single subset of $P(\Delta)$, namely

$$\{ \ (v_1, \ldots, v_N) \in P(\Delta) \ \mid \ \varphi(v_1, \ldots, v_N) \ \} \ .$$

In the rest of the paper we work with one concrete language obtained as follows: Each parameter range is a finite set, which is explicitly listed in the declaration. Each primitive constraint has one of the two forms

$$\begin{aligned} q_i &= c_i \\ q_i &\neq c_i \end{aligned}$$

where q_i is one of the declared parameters, and c_i is one of the values in the range of q_i; it is obvious how these constraints evaluate to true or false. Each primitive criterion has one of the three forms

$$\langle\ q_i\ =\ c_i\ \rangle$$
$$\langle\ q_i\ \neq\ c_i\ \rangle$$
$$\mathsf{ANY\_TEST}$$

where q_i is one of the declared parameters and c_i is one of the values in the range of q_i. The values $\mathsf{S}(\langle q_i = c_i\rangle)$ and $\mathsf{S}(\langle q_i \neq c_i\rangle)$ are defined by the convention at the beginning of the previous paragraph. The value $\mathsf{S}(\mathsf{ANY\_TEST})$ contains only the set $P(\Delta)$ itself.

The present definition of $\mathsf{S}(\Gamma)$ differs slightly from the definition of the "pile assigned to Γ" in the previous design of the language [11]; namely, we do not require that $\emptyset \in \mathsf{S}(\Gamma)$ and $P(\Gamma) \in \mathsf{S}(\Gamma)$. We find the present definition technically more convenient.

Using these primitive criteria and the $\uplus$ and $\otimes$ operations, the test designer can write down many other useful criteria. In particular, it is possible to specify that a particular vector $(v_1, \ldots, v_N)$ of parameter values $v_i \in Q_i$ must be included in the selected test set. For example, to ensure that the vector in which

$$\begin{aligned}
\textbf{separator_1} = \textbf{separator_2} &= \text{"}\texttt{/}\text{"} \\
\textbf{string_1} &= \text{"}\texttt{abcd}\text{"} \\
\textbf{string_2} &= \text{"}\texttt{ab}\text{"} \\
\textbf{string_1_occurs} &= \mathsf{true}
\end{aligned}$$

is in the selected set, the test designer would use the criterion

$$\langle\ \textbf{separator\_1} = \text{"}\texttt{/}\text{"}\ \rangle\ \otimes\ \langle\ \textbf{separator\_2} = \text{"}\texttt{/}\text{"}\ \rangle\ \otimes\ \langle\ \textbf{string\_1} = \text{"}\texttt{abcd}\text{"}\ \rangle\ \otimes$$
$$\langle\ \textbf{string\_2} = \text{"}\texttt{ab}\text{"}\ \rangle\ \otimes\ \langle\ \textbf{string\_1\_occurs} = \mathsf{true}\ \rangle\ .$$

The criteria EACH and $\mathsf{EXHAUSTIVE}$, which were informally described in the previous section, can also be constructed using $\uplus$ and $\otimes$. The general definition is as follows: Let Q_i

be the range of the parameter q_i. If $Y \subseteq Q_i$ then define

$$\mathsf{EACH}(\ q_i\ :\ Y\ )\ =\ \biguplus_{a \in Y} \langle q_i = a \rangle\ .$$

The criterion specifies that each value in Y must be tested (as long as there is at least one point in $D(\psi)$ with that value of q_i).

For any sequence $q_{i_1}, q_{i_2}, \ldots, q_{i_m}$ of parameters, define

$$\mathsf{EXHAUSTIVE}(q_{i_1}, q_{i_2}, \ldots, q_{i_m})\ =\ \bigotimes_{j=1}^{m} \mathsf{EACH}(\ q_{i_j}\ :\ Q_{i_j}\ )\ .$$

This specifies that all the combinations of values of $q_{i_1}, q_{i_2}, \ldots, q_{i_m}$ allowed by the constraint must be tested.

The criteria $\mathsf{EXHAUSTIVE}(q_1, \ldots, q_N)$ and $\mathsf{ANY\_TEST}$ are at opposite ends of the scale ordered by $\sqsubseteq$. Only the set $D(\psi)$ itself is adequate for $\mathsf{EXHAUSTIVE}(q_1, \ldots, q_N)$. If $D(\psi) \neq \emptyset$, any nonempty subset of $D(\psi)$ is adequate for $\mathsf{ANY\_TEST}$.

4 Worst-case complexity of two test selection problems

In this section we work with the concrete language from Section 3.3, and we consider algorithmic aspects of the criteria specified in the language: Given one such criterion, how difficult is it to find an adequate test set that is in some sense "small"?

4.1 Two basic problems

Let $I = (\Delta, \psi, \Gamma)$ be an instance of the test selection problem, and let T be an adequate test set for I. Say that T is a **minimum adequate test set** if no set of cardinality smaller than $|T|$ is adequate. Say that T is a **minimal adequate test set** if no proper subset of T is adequate.

We are interested in algorithms for two problems:

The Minimum Adequate Set Search Problem (MumAS)

Input: An instance $I = (\Delta, \psi, \Gamma)$.

Output: A minimum adequate test set for I.

The Minimal Adequate Set Search Problem (MalAS)

Input: An instance $I = (\Delta, \psi, \Gamma)$.

Output: A minimal adequate test set for I.

The **size of the instance** $I = (\Delta, \psi, \Gamma)$, denoted $|I|$, is the total length of the declarations in Δ and of the expressions ψ and Γ. Often the cardinality of the set $P(\Delta)$ is exponential in the number of parameters in Δ. For example, if each parameter range Q_i, $1 \leq i \leq N$, consists of two values then the cardinality of $P(\Delta)$ is 2^N. Thus the cardinality of a minimum or minimal adequate test set T may be exponential in $|I|$; in that case no algorithm that outputs T can execute in time polynomial in $|I|$. We shall therefore measure the execution time of such algorithms in terms of $|I| + |T|$. Thus a *polynomial-time algorithm* for MumAS or MalAS is an algorithm whose worst-case execution time is bounded by a polynomial function of $|I| + |T|$.

We shall identify two obstacles on the path toward efficient algorithms for MumAS and MalAS. One obstacle, related to the boolean satisfiability problem, applies to both MumAS and MalAS (section 4.2); the other, related to graph colorability, applies only to MumAS (section 4.3).

4.2 Connections with boolean satisfiability

For classifying problems as NP-complete, NP-hard, etc., we use the terminology of Garey and Johnson [4]. MumAS and MalAS are *search* problems ([4], p. 110). The following *decision* problem will be useful in our analysis of the complexity of MumAS and MalAS.

The Empty Adequate Set Problem (EA)

Input: An instance I.

Question: Is the empty set adequate for I?

Denote by $\bigcup \mathsf{S}(\Gamma)$ the union of all sets in $\mathsf{S}(\Gamma)$. The empty set is adequate for $I = (\Delta, \psi, \Gamma)$ if and only if $D(\psi) \cap \bigcup \mathsf{S}(\Gamma) = \emptyset$.

It is not difficult to prove that EA is in co-NP. However, we are more interested in proving that EA is NP-hard; we now prove the NP-hardness of EA, by reduction from the

boolean satisfiability problem.

Theorem 4.1 *The problem* EA *is NP-hard, even if the input* $I = (\Delta, \psi, \Gamma)$ *is such that*

1. $\psi = \mathsf{true}$, *or*
2. $\Gamma = \mathsf{ANY\_TEST}$.

Proof. By reduction from $\mathsf{3SAT}$ ([4], p. 46). Let C be an instance of $\mathsf{3SAT}$. We construct an instance I such that C is satisfiable if and only if $\emptyset$ is not adequate for I.

Let $C = \{c_1, c_2, \ldots, c_m\}$ be a set of clauses on a finite set U of boolean variables, such that

$$c_j = a_{j1} \vee a_{j2} \vee a_{j3}$$

for $1 \leq j \leq m$. Each literal a_{jk} is either a variable u in U or its negation $\overline{u}$. Let Δ be the declarations

$$u : \{\ \mathsf{true}, \mathsf{false}\ \}$$

for u in U.

For $\psi = \mathsf{true}$, the empty set is adequate if and only if $\bigcup \mathsf{S}(\Gamma) = \emptyset$. Define

$$\Gamma = \bigotimes_{j=1}^{m} (\Gamma_{j1} \uplus \Gamma_{j2} \uplus \Gamma_{j3})$$

where

$$\Gamma_{jk} \quad = \quad \begin{cases} \langle u = \mathsf{true} \rangle & \text{if } a_{jk} = u \\ \langle u = \mathsf{false} \rangle & \text{if } a_{jk} = \overline{u} \end{cases}$$

for $k = 1, 2, 3$, and define $I = (\Delta, \mathsf{true}, \Gamma)$. Then C is satisfiable if and only if $\bigcup \mathsf{S}(\Gamma) \neq \emptyset$.

For $\Gamma = \mathsf{ANY\_TEST}$, the empty set is adequate if and only if $D(\psi) = \emptyset$. Define

$$\psi = \bigwedge_{j=1}^{m} (\psi_{j1} \vee \psi_{j2} \vee \psi_{j3})$$

where

$$\psi_{jk} = \begin{cases} u = \mathsf{true} & \text{if } a_{jk} = u \\ u = \mathsf{false} & \text{if } a_{jk} = \overline{u} \end{cases}$$

and define $I = (\Delta, \psi, \mathsf{ANY\_TEST})$. Then C is satisfiable if and only if $D(\psi) \neq \emptyset$. □

The following lemma shows that any lower bound for the execution time complexity of EA implies a lower bound for MumAS and MalAS.

Lemma 4.2 *Let w be an integer function of an integer variable such that the value $w(i)$ for any integer i can be computed in $O(w(i))$ steps. If there exists an algorithm for* MumAS *or* MalAS *that for every input I produces an output T in at most $w(|I| + |T|)$ steps, then there exists an algorithm that solves* EA *for every input I in $O(w(|I|))$ steps.*

Proof. To solve EA on input I, compute $w(|I|)$ and execute the algorithm for MumAS (or MalAS) on input I for at most $w(|I|)$ steps. The answer to the question in EA is "yes" if the algorithm terminates with output $T = \emptyset$. The answer is "no" if the algorithm terminates with output $T \neq \emptyset$ or does not terminate in $w(|I|)$ steps. □

Theorem 4.3 *If $P \neq NP$ then neither* MumAS *nor* MalAS *is solvable by a polynomial-time algorithm, even in cases 1 and 2 in Theorem 4.1.*

Proof. Apply Theorem 4.1 and Lemma 4.2. □

4.3 Connections with graph colorability

We have identified one reason why MumAS and MalAS are difficult: ψ and Γ may encode arbitrary boolean expressions, and thus any algorithm for MumAS or MalAS can be used to construct an algorithm for 3SAT. It is therefore natural to ask whether MumAS and MalAS become easier when ψ and Γ belong to a smaller class of expressions.

We start with a simple such class, the criteria in $\uplus\otimes=$ form. The $\uplus\otimes=$ **form** of a test selection criterion is

$$\biguplus_{j=1}^{m} \bigotimes_{k=1}^{n_j} \Gamma_{jk} \tag{6}$$

where Γ_{jk} are primitive criteria of the form $\langle q_i = c_i \rangle$. Define an instance $I = (\Delta, \psi, \Gamma)$ to be **simple** if $\psi = \mathsf{true}$ and Γ is in $\uplus\otimes=$ form. In the next section we shall see that the problem MaIAS for simple instances is solvable by a polynomial-time algorithm. In contrast, MumAS for simple instances is NP-hard, as will be established in Theorem 4.8. The following decision problem will be used in the proof.

The Minimum Adequate Set Problem for Simple Instances (MASI)

Input: A simple instance I and an integer K.

Question: Is there a set T adequate for I such that $|T| \leq K$?

We are going to show that MASI is equivalent to GRAPH K-COLORABILITY ([4], p. 191).

Let S be a set of sets. The **intersection graph** of S is the graph $G = (\mathsf{S}, E)$ in which the set of vertices is S and the set of edges is

$$E = \{\ \{X, Y\}\ \mid\ X, Y \in \mathsf{S},\ X \neq Y \ \text{ and } \ X \cap Y \neq \emptyset\ \}\ .$$

When $G = (V, E)$ is a graph, the **complement** of G is the graph $\overline{G} = (V, \overline{E})$, where

$$\overline{E} = \{\ \{x, y\}\ \mid\ x, y \in V,\ x \neq y \ \text{ and } \ \{x, y\} \notin E\ \}\ .$$

The proof of the following simple lemma is left to the reader. Note that the lemma would not be true if we admitted primitive criteria of the form $\langle q_i \neq c_i \rangle$.

Lemma 4.4 *Let $I = (\Delta, \mathsf{true}, \Gamma)$ be a simple instance, and let $\mathsf{S}_0 \subseteq \mathsf{S}(\Gamma)$. If $X \cap Y \neq \emptyset$ for all $X, Y \in \mathsf{S}_0$ then $\bigcap\{X | X \in \mathsf{S}_0\} \neq \emptyset$.* □

By the lemma, a set $\mathsf{S}_0 \subseteq \mathsf{S}(\Gamma)$ forms a clique in the intersection graph of $\mathsf{S}(\Gamma)$ if and only if $\bigcap\{X | X \in \mathsf{S}_0\} \neq \emptyset$.

Proposition 4.5 *Let $I = (\Delta, \mathsf{true}, \Gamma)$ be a simple instance, let $\overline{G}$ be the complement of the intersection graph of $\mathsf{S}(\Gamma) \setminus \{\emptyset\}$, and let K be an integer. The graph $\overline{G}$ is K-colorable if and only if there exists a set T adequate for I such that $|T| \leq K$.*

Proof. Assume $\overline{G} = (\mathsf{S}(\Gamma) \setminus \{\emptyset\}, \overline{E})$ is K-colorable. This means that there exists a mapping $f : \mathsf{S}(\Gamma) \setminus \{\emptyset\} \rightarrow \{1, 2, \ldots, K\}$ such that $f(X) \neq f(Y)$ when $\{X, Y\} \in \overline{E}$. Define

$$\mathsf{S}_j = \{ \ X \in \mathsf{S}(\Gamma) \setminus \{\emptyset\} \mid f(X) = j \ \}$$

for $j = 1, 2, \ldots, K$. From the definition of the intersection graph we get that if $X, Y \in \mathsf{S}_j$ then $X \cap Y \neq \emptyset$; by Lemma 4.4 we have $\bigcap\{X|X \in \mathsf{S}_j\} \neq \emptyset$. Form a set T by choosing one point in each $\bigcap\{X|X \in \mathsf{S}_j\}$, $j = 1, 2, \ldots, K$. Thus $|T| \leq K$ and T intersects each nonempty $X \in \mathsf{S}(\Gamma)$, which means that T is adequate for I.

Conversely, assume that there exists a set T adequate for I such that $|T| \leq K$. Write $T = \{d_1, d_2, \ldots, d_K\}$ and define a K-coloring $f : \mathsf{S}(\Gamma) \setminus \{\emptyset\} \rightarrow \{1, 2, \ldots, K\}$ of $\overline{G}$ by

$$f(X) = \min\{ \ j \mid d_j \in X \ \} \ .$$

Since T is adequate, f is defined for each $X \in \mathsf{S}(\Gamma) \setminus \{\emptyset\}$. If $f(X) = f(Y) = j$ then $d_j \in X \cap Y$, hence $X \cap Y \neq \emptyset$, hence $\{X, Y\}$ is not an edge in $\overline{G}$. Thus f is a coloring of $\overline{G}$.

□

Proposition 4.6 *For each graph $G = (V, E)$ there exists a simple instance $I = (\Delta, \mathsf{true}, \Gamma)$ such that the intersection graph of $\mathsf{S}(\Gamma)$ is (isomorphic to) G. The declarations Δ consist of one boolean parameter for each vertex in V.*

Proof. Let V consist of N vertices, $V = \{x_1, x_2, \ldots, x_N\}$. Let Δ be the declarations

$$q_i : \{\mathsf{true}, \mathsf{false}\}$$

for $i = 1, 2, \ldots, N$. Define

$$H(i) = \{ \ j \in \{1, 2, \ldots, N\} \ \mid \ j \neq i \ \text{ and } \ \{x_i, x_j\} \notin E \ \} \ ,$$

$$\Gamma \;=\; \biguplus_{i=1}^{N} \;(\; \langle q_i = \mathsf{true}\rangle \otimes \bigotimes_{j\in H(i)} \langle q_j = \mathsf{false}\rangle \;)\;\; .$$

Then $\mathsf{S}(\Gamma)$ consists of the sets

$$X_i = \{\; (v_1,\ldots,v_N) \in P(\Delta) \;\mid\; v_i = \mathsf{true} \;\; \text{and} \quad \forall j \in H(i) \;:\; v_j = \mathsf{false} \;\}$$

for $i = 1,2,\ldots,N$. We have $X_i \neq X_j$ for $i \neq j$, the sets X_i are nonempty, and the mapping $x_i \mapsto X_i$ is an isomorphism between G and the intersection graph of $\mathsf{S}(\Gamma)$. □

By Propositions 4.5 and 4.6, GRAPH K-COLORABILITY and MASI are polynomially equivalent. From known results for GRAPH K-COLORABILITY ([4], p. 191) we obtain the following result for MASI.

Theorem 4.7 *The problem* MASI *is NP-complete, even for $K = 3$.* □

It remains to transform MASI into MumAS. The only potential complication is that "polynomial" means "polynomial in the size of input" for MASI and "polynomial in the size of input *and output*" for MumAS. However, if the input instance I is simple and the output set T is minimum then $|T|$ is bounded by $|I|$. Indeed, for criterion (6) there exists an adequate test set of cardinality at most m, which means that the cardinality of the minimum set T is also bounded by m. Thus Theorem 4.7 yields the following result for MumAS.

Theorem 4.8 *If $P \neq NP$ then* MumAS *is not solvable by a polynomial-time algorithm, even for simple instances.* □

In view of Theorem 4.8, we are not likely to find a polynomial-time algorithm for MumAS. It is still possible that there is an algorithm for MumAS that is efficient in some other sense, but we have not been able to find any such algorithm. However, in the next section we present a practical algorithm for MaIAS.

The transformation in Propositions 4.5 and 4.6 yields more than results for MASI and MumAS. For example, if we had an algorithm that for every simple instance would find an adequate test set whose cardinality is within the factor $(1+\varepsilon)$ of the minimum, then we would also have an algorithm to color any graph with the number of colors within the factor $(1+\varepsilon)$ of the minimum. No such polynomial-time algorithm is presently known for any fixed constant ε.

By virtue of Proposition 4.5, any algorithm for graph coloring can be transformed into an algorithm for constructing adequate test sets for simple instances; when the graph coloring uses the minimum number of colors, the adequate test set is minimum. Many heuristic algorithms for graph coloring have been studied; see e.g. [12, 13] and the references therein. However, we are interested in the problems MumAS and MaIAS, rather than MASI; the restriction to simple instances is severe. We have already noted that Lemma 4.4 does not hold if primitive criteria $\langle q_i \neq c_i \rangle$ are allowed. Moreover, if ψ is a general constraint then Lemma 4.4 may fail for the sets in $\mathsf{S}(\Gamma)$ restricted to the domain $D(\psi)$.

5 Algorithms for finding minimal adequate test sets

5.1 An algorithm for normalized instances

In this section we concentrate on the problem MaIAS defined in Section 4.1. We start with an efficient algorithm for the input instances $I = (\Delta, \psi, \Gamma)$ in which ψ and Γ belong to a certain restricted class of expressions. Afterwards we show how to use the algorithm for general instances.

The test selection criterion

$$\biguplus_{j=1}^{r} \bigotimes_{k=1}^{s_j} \Gamma_{jk} \quad , \tag{7}$$

where Γ_{jk} are primitive criteria, is said to be in the $\uplus\otimes$ form. The constraint

$$\bigvee_{j=1}^{m} \bigwedge_{k=1}^{n_j} \psi_{jk} \quad , \tag{8}$$

where ψ_{jk} are primitive constraints, is said to be in the $\vee\wedge$ form. (This is also called the disjunctive normal form.)

An instance $I = (\Delta, \psi, \Gamma)$ is **normalized** if ψ is in the $\vee\wedge$ form and Γ in the $\uplus\otimes$ form.

Let Δ be a fixed set of parameter declarations $q_i : Q_i,\ \ i = 1, 2, \ldots, N$. We say that a set $X \subseteq P(\Delta)$ is a **subcube** if it is in the form $\prod_{i=1}^{N} R_i$ where $R_i \subseteq Q_i$. For our concrete language of Section 3.3, every $X \in \mathsf{S}(\Gamma)$ is a subcube. When the criterion Γ is in the $\uplus\otimes$ form, it is easy to compute the set $\mathsf{S}(\Gamma)$: The subcubes in $\mathsf{S}(\Gamma)$ correspond to the terms $\bigotimes_k \Gamma_{jk}$ in (7). Similarly, every term $\bigwedge_k \psi_{jk}$ in (8) defines the subcube $D(\bigwedge_k \psi_{jk})$, and $D(\bigvee_j \bigwedge_k\ \psi_{jk}) = \bigcup_j\ D(\bigwedge_k\ \psi_{jk})$.

The algorithm in Figure 2 constructs a minimal set adequate for a given normalized instance. The input for the algorithm consists of two sets of subcubes: the set $S = \mathsf{S}(\Gamma)$, and the set

$$C = \{\ D(\textstyle\bigwedge_k\ \psi_{jk}) \mid j = 1, 2, \ldots, m\ \}$$

for the constraint (8). When the algorithm terminates, the set variable T contains a minimal adequate set.

In the program for the algorithm, **forall** denotes iteration over all elements of a set in some arbitrary order. The values of the data type "point" are the elements of $P(\Delta)$. The function call Find_point(X, C) finds a point in the set $X \cap \bigcup C$; if the set is empty, the function returns NIL.

For each $t \in T$, the variable $contains(t)$ stores a set of subcubes; a subcube $X \in S$ belongs to $contains(t)$ if and only if $t \in X$. For each $X \in S$, the variable $count(X)$ stores the cardinality of $X \cap T$.

The algorithm works in two phases: The first phase finds an adequate test set, and the second phase trims the set to make it minimal.

When sets are represented as arrays or linked lists, adding one element takes constant time, and iterating through a **forall** loop adds only constant time per iteration. The deletion operation on the last line of the program is implemented by marking the element as deleted; that also takes only constant time.

When points and subcubes are represented as sorted lists of primitive constraints, the function Find_point(X, C) and the test "**if** $t \in Y$" are implemented by a single pass through

```
inputs
    S   : set of subcube
    C   : set of subcube

variables
    T             : set of point
    contains(t)   : set of subcube,   for t ∈ T
    count(X)      : integer,   for X ∈ S

initially
    T = ∅
    count(X) = 0,   for X ∈ S

program
    forall  X ∈ S  do
        if  count(X) = 0  then
            t := Find_point( X, C )
            if  t ≠ NIL  then
                T := T ∪ {t}
                forall  Y ∈ S  do
                    if  t ∈ Y  then
                        contains(t) := contains(t) ∪ {Y}
                        count(Y) := count(Y) + 1

    forall  t ∈ T  do
        if  max( count(Y), Y ∈ contains(t) ) ≥ 2  then
            forall  Y ∈ contains(t)  do
                count(Y) := count(Y) − 1
            T := T \ {t}
```

Figure 2: Algorithm for MaIAS

the lists representing the two arguments. Adding it all up, we get the bound $O(|I|^2)$ for the total execution time of the algorithm on any input instance I. We summarize our analysis in a theorem.

Theorem 5.1 *There is an algorithm to solve the problem* MaIAS *for any normalized in-*

stance I in time $O(|I|^2)$. $\square$

5.2 The cost of normalization

Requiring input instances to be normalized would be inconvenient to the users. For example:

- The constraint is often naturally specified in the conjunctive, rather than disjunctive, normal form.
- The user should be able to take any two criteria and combine them by means of $\otimes$. The resulting criterion is not in the $\uplus\otimes$ form.

Therefore our design allows users to specify the instance in the general form defined in Section 3. The instance is automatically converted into an equivalent normalized form before the algorithm in Figure 2 is applied.

The normalization is easy to implement. The well-known procedure transforms boolean expressions into $\vee\wedge$ form by repeatedly replacing a conjuction of disjunctions by an equivalent disjunction of conjunctions. By virtue of the distributive law for $\uplus$ and $\otimes$, the same procedure works for the test selection criteria built using $\uplus$ and $\otimes$.

However, the user should understand that the normalization may in some cases be expensive, in terms of execution time. In the worst case, the execution time is exponential in the size of the original expression. We shall now discuss the implications of the normalization cost, separately for the constraint expression ψ and for the criterion expression Γ.

For ψ, the exponential increase of the execution time of the normalization procedure is more common and more serious than for Γ. A large instance $I = (\Delta, \psi, \Gamma)$ of the test selection problem is typically obtained by putting together several instances $I_j = (\Delta_j, \psi_j, \Gamma_j)$ with disjoint sets of parameters. It is then natural to take $\psi = \bigwedge_j \psi_j$. If m independent constraints are put together to form

$$\psi = \bigwedge_{j=1}^{m} (\psi_{j1} \vee \psi_{j2})$$

then the equivalent $\vee\wedge$ form of ψ has 2^m terms. Thus in this case the total execution time is at least proportional to 2^m, even if the test set produced at the end is very small.

The cost of normalizing the criterion Γ is less critical; in most cases large criteria lead to large test sets. However, "in most cases" does not mean "always", as the following example shows:

Example. Let Δ consist of N declarations $q_i : \{0,1\}$, $1 \leq i \leq N$. Consider the criterion

$$\bigotimes_{j=1}^{N} \biguplus_{i=1}^{N} \langle q_i = 0 \rangle \quad . \tag{9}$$

The equivalent $\uplus\otimes$ form is

$$\biguplus_{A} \bigotimes_{i \in A} \langle q_i = 0 \rangle \quad , \tag{10}$$

where A runs through all nonempty subsets of $\{1, 2, \ldots, N\}$. The only minimal adequate test set is $T = \{(0, 0, \ldots, 0)\}$, of cardinality 1. In transforming (9) to (10) the algorithm generates all the $2^N - 1$ expressions

$$\bigotimes_{i \in A} \langle q_i = 0 \rangle \quad ,$$

where $\emptyset \neq A \subseteq \{1, 2, \ldots, N\}$. $\square$

Nevertheless, we conjecture that, in most practical situations, if the test designer specifies a selection criterion whose equivalent $\uplus\otimes$ form is very large, then every adequate test set will also be very large. In such cases long execution time (at least proportional to the size of the produced test set) cannot be avoided.

In the next section we shall show that for the instance I built by combining independent instances I_j, we can solve the problem MaIAS separately for each I_j and then put the solutions together to produce a test set adequate for I. We shall also describe an algorithm for decomposing instances into independent components. We expect that for most instances of the test selection problem arising in practice the decomposition method will avoid the exponential cost of normalization.

5.3 Decomposition of instances

When the test designer constructs a large instance of the test selection problem, it is likely that the instance is built from subproblems that are in some sense independent. Now we show how such structure can be exploited to construct minimal adequate test sets.

Definition. Two instances $I_j = (\Delta_j, \psi_j, \Gamma_j)$, $j = 1, 2$, are **independent** if no parameter occurs in both Δ_1 and Δ_2.

Definition. Let $I_j = (\Delta_j, \psi_j, \Gamma_j)$, $j = 1, 2$, be two independent instances. Let $\Delta = \Delta_1 \cup \Delta_2$. Define two instances

$$\begin{aligned} I_1[\wedge\otimes]I_2 &= (\Delta, \psi_1 \wedge \psi_2, \Gamma_1 \otimes \Gamma_2) \\ I_1[\wedge\uplus]I_2 &= (\Delta, \psi_1 \wedge \psi_2, \Gamma_1 \uplus \Gamma_2) \end{aligned}$$

When α is $\wedge\otimes$ or $\wedge\uplus$, we say that I_1 and I_2 form an *independent α-decomposition* (or simply a *decomposition*) of $I_1[\alpha]I_2$.

We now construct adequate test sets for $I_1[\wedge\otimes]I_2$ and $I_1[\wedge\uplus]I_2$ from adequate test sets for I_1 and I_2. For two nonempty sets T_1 and T_2 such that $|T_1| = m$, $|T_2| = n$, define the set $T_1 \parallel T_2 \subseteq T_1 \times T_2$ as follows: Let $T_1 = \{r_1, r_2, \ldots, r_m\}$, $T_2 = \{s_1, s_2, \ldots, s_n\}$, and

$$T_1 \parallel T_2 = \begin{cases} \{\ (r_1, s_1), (r_2, s_2), \ldots, (r_m, s_m)\ \} & \text{if } m = n \\ \{\ (r_1, s_1), (r_2, s_2), \ldots, (r_n, s_n), (r_{n+1}, s_n), \ldots, (r_m, s_n)\ \} & \text{if } m > n \\ \{\ (r_1, s_1), (r_2, s_2), \ldots, (r_m, s_m), (r_m, s_{m+1}), \ldots, (r_m, s_n)\ \} & \text{if } m < n \end{cases}$$

Thus the definition of $T_1 \parallel T_2$ depends on the order in which we number the elements of T_1 and T_2; we assume that one such order is chosen arbitrarily.

Definition. Let $I_j = (\Delta_j, \psi_j, \Gamma_j)$, $j = 1, 2$, be two independent instances such that $D(\psi_j) \neq \emptyset$, and let $T_j \subseteq D(\psi_j)$ for $j = 1, 2$. Define

$$\begin{aligned} T_1[\wedge\otimes]T_2 &= T_1 \times T_2 \\ T_1[\wedge\uplus]T_2 &= \begin{cases} \emptyset & \text{if } T_1 = \emptyset = T_2 \\ T_1 \times \{s_2\} & \text{if } T_1 \neq \emptyset = T_2 \\ \{s_1\} \times T_2 & \text{if } T_1 = \emptyset \neq T_2 \\ T_1 \parallel T_2 & \text{if } T_1 \neq \emptyset \neq T_2 \end{cases} \end{aligned}$$

where $s_j \in D(\psi_j)$, $j = 1, 2$, are some arbitrarily chosen elements.

Theorem 5.2 *Let $I_j = (\Delta_j, \psi_j, \Gamma_j)$, $j = 1, 2$, be two independent instances such that $D(\psi_j) \neq \emptyset$, and let α be $\wedge\otimes$ or $\wedge\uplus$. If T_j is an adequate test set for I_j, $j = 1, 2$, then $T_1[\alpha]T_2$ is an adequate test set for $I_1[\alpha]I_2$. If T_j is a minimal adequate test set for I_j, $j = 1, 2$, then $T_1[\alpha]T_2$ is a minimal adequate test set for $I_1[\alpha]I_2$.*

Proof. Let $\Delta = \Delta_1 \cup \Delta_2$, $P_1 = P(\Delta_1)$, $P_2 = P(\Delta_2)$, $P = P(\Delta)$, and $\psi = \psi_1 \wedge \psi_2$. Thus

$$\begin{aligned}
P &= P_1 \times P_2 \\
D(\Delta, \psi) &= D(\Delta_1, \psi_1) \times D(\Delta_2, \psi_2) \\
\mathsf{S}(\Delta, \Gamma_1) &= \{\, X_1 \times P_2 \mid X_1 \in \mathsf{S}(\Delta_1, \Gamma_1) \,\} \\
\mathsf{S}(\Delta, \Gamma_2) &= \{\, P_1 \times X_2 \mid X_2 \in \mathsf{S}(\Delta_2, \Gamma_2) \,\} \\
\mathsf{S}(\Delta, \Gamma_1 \uplus \Gamma_2) &= \mathsf{S}(\Delta, \Gamma_1) \,\cup\, \mathsf{S}(\Delta, \Gamma_2) \\
\mathsf{S}(\Delta, \Gamma_1 \otimes \Gamma_2) &= \{\, X_1 \times X_2 \mid X_1 \in \mathsf{S}(\Delta_1, \Gamma_1)\,,\ X_2 \in \mathsf{S}(\Delta_2, \Gamma_2) \,\}
\end{aligned}$$

Let T_j be an adequate test set for I_j, $j = 1, 2$; that is, $T_j \cap X \neq \emptyset$ whenever $X \in \mathsf{S}(\Delta_j, \Gamma_j)$ and $X \cap D(\Delta_j, \psi_j) \neq \emptyset$. Let $T = T_1[\alpha]T_2$.

For $\alpha = \wedge\otimes$, if $X_1 \times X_2 \in \mathsf{S}(\Delta, \Gamma_1 \otimes \Gamma_2)$ and $(X_1 \times X_2) \cap D(\Delta, \psi) \neq \emptyset$ then $X_j \in \mathsf{S}(\Delta_j, \Gamma_j)$, $X_j \cap D(\Delta_j, \psi_j) \neq \emptyset$. Therefore $X_j \cap T_j \neq \emptyset$ and $(X_1 \times X_2) \cap T \neq \emptyset$.

For $\alpha = \wedge\uplus$, if $X_1 \times P_2 \in \mathsf{S}(\Delta, \Gamma_1)$ and $(X_1 \times P_2) \cap D(\Delta, \psi) \neq \emptyset$ then $X_1 \in \mathsf{S}(\Delta_1, \Gamma_1)$, $X_1 \cap D(\Delta_1, \psi_1) \neq \emptyset$. Therefore $X_1 \cap T_1 \neq \emptyset$ and $(X_1 \times P_2) \cap T \neq \emptyset$. The argument for $P_1 \times X_2 \in \mathsf{S}(\Delta, \Gamma_2)$ is symmetrical.

Now let T_j be a minimal adequate test set for I_j, $j = 1, 2$, and let $T = T_1[\alpha]T_2$.

Let $\alpha = \wedge\otimes$. To prove that T is minimal, take any $(t_1, t_2) \in T$. Since T_j is minimal, there is $X_j \in \mathsf{S}(\Delta_j, \psi_j)$ such that $X_j \cap D(\Delta_j, \psi_j) \neq \emptyset$ and $X_j \cap (T_j \setminus \{t_j\}) = \emptyset$. For $X = X_1 \times X_2$ we have $X \cap D(\Delta, \psi) \neq \emptyset$ and $X \cap (T \setminus \{(t_1, t_2)\}) = \emptyset$. Therefore $T \setminus \{(t_1, t_2)\}$ is not adequate. Thus T is minimal.

Let $\alpha = \wedge\uplus$. If $T_1 = T_2 = \emptyset$ then $T = \emptyset$, hence T is minimal. Now assume, without loss of generality, that $|T_1| \geq |T_2|$ and $T_1 \neq \emptyset$. Then for every $t_1 \in T_1$ there exists exactly one $t_2 \in D(\psi_2)$ such that $(t_1, t_2) \in T$. To prove that T is minimal, take any $(t_1, t_2) \in T$. Since T_1 is minimal, there is $X_1 \in \mathsf{S}(\Delta_1, \psi_1)$ such that $X_1 \cap D(\Delta_1, \psi_1) \neq \emptyset$ and $X_1 \cap (T_1 \setminus \{t_1\}) = \emptyset$. For $X = X_1 \times P_2$ we have $X \cap D(\Delta, \psi) \neq \emptyset$ and $X \cap (T \setminus \{(t_1, t_2)\}) = \emptyset$. Therefore

$T \setminus \{(t_1, t_2)\}$ is not adequate. Thus T is minimal. $\square$

One can prove that if T_1 and T_2 are *minimum* adequate then $T_1[\wedge\uplus]T_2$ is also minimum. However, the same is not true for $T_1[\wedge\otimes]T_2$, as the following example shows:

Example. Define two instances $I_j = (\Delta_j, \psi_j, \Gamma_j)$, $j = 1, 2$: The declaration Δ_j is

$$x_j \ : \ \{1, 2, 3\} \ ,$$

there is no constraint (i.e. $\psi_j = \mathsf{true}$), and the criterion Γ_j is

$$\langle x_j \neq 1 \rangle \uplus \langle x_j \neq 2 \rangle \uplus \langle x_j \neq 3 \rangle \ .$$

If T_1 and T_2 are minimum adequate sets for I_1 and I_2 then $|T_1| = |T_2| = 2$, hence $|T_1 \times T_2| = 4$. However, the three-element set $\{(1,1), (2,2), (3,3)\}$ is adequate for $I_1[\wedge\otimes]I_2$. $\square$

To utilize Theorem 5.2 in constructing minimal test sets, we simply add the operations $[\wedge\otimes]$ and $[\wedge\uplus]$ on instances to the language. The test designer may then specify a large instance as a combination of smaller components, using $[\wedge\otimes]$ and $[\wedge\uplus]$. In fact, if the language has appropriate scoping rules for the names of parameters then we need not require that the parameter names in the component instances be different.

Now we describe a simple algorithm for discovering a decomposition into independent instances, when the decomposition is not explicitly specified by the test designer. The algorithm works on the instances $I = (\Delta, \psi, \Gamma)$ in which ψ has the form $\bigwedge_k \psi_k$. The algorithm groups some terms ψ_k and some subexpressions of Γ together, but does not attempt to use distributive laws to transform the expressions ψ and Γ.

Consider an instance $I = (\Delta, \psi, \Gamma)$ in which Δ consists of declarations $q_i : Q_i$, $i = 1, 2, \ldots, N$. **Subexpressions** (often called **well-formed subexpressions**) of Γ correspond to subtrees of the parse tree of Γ. For $i = 1, 2, \ldots, N$, let $\Gamma(i)$ be the smallest subexpression of Γ that contains all occurrences of q_i in Γ; in the parse tree of Γ, $\Gamma(i)$ corresponds to the smallest subtree containing all the leaves labeled $\langle q_i = c \rangle$ and $\langle q_i \neq c \rangle$.

Define two binary relations W_ψ and W_Γ on the set $\{1, 2, \ldots, N\}$:

- $i\ W_\psi\ j$ if i and j occur in ψ_k, for some k;
- $i\ W_\Gamma\ j$ if j occurs in $\Gamma(i)$.

Let W be the finest equivalence relation on $\{1, 2, \ldots, N\}$ such that $W \supseteq W_\psi \cup W_\Gamma$. Computing W is a straightforward application of the transitive-closure algorithm ([1], p. 199). Now each equivalence class B of W determines a subset Δ_B of the declarations Δ; the subsets Δ_B are pairwise disjoint. By the construction of W we have

$$\psi = \bigwedge_B \bigwedge_{k \in B} \psi_k$$

where $\bigwedge_B$ is the conjunction over the equivalence classes B of W. Each equivalence class B determines a subexpression Γ_B. The expression Γ is formed from Γ_B by means of $\uplus$ and $\otimes$. Thus we have decomposed I into independent instances I_B, from which I is formed by means of $[\wedge\uplus]$ and $[\wedge\otimes]$.

It is of course possible that $i\,W\,j$ for all $i, j \in \{1, 2, \ldots, N\}$. In that case this simple approach to decomposition does not help. However, in those cases where I has been formed by combining several independent instances using $[\wedge\uplus]$ and $[\wedge\otimes]$, the algorithm will lead back at least to the original independent instances, and it may even discover a decomposition into smaller instances.

5.4 Generalized decomposition

In analogy to the operations $[\wedge\otimes]$ and $[\wedge\uplus]$, we can also define

$$\begin{aligned} I_1[\vee\otimes]I_2 &= (\Delta_1 \cup \Delta_2, \psi_1 \vee \psi_2, \Gamma_1 \otimes \Gamma_2) \\ I_1[\vee\uplus]I_2 &= (\Delta_1 \cup \Delta_2, \psi_1 \vee \psi_2, \Gamma_1 \uplus \Gamma_2) \end{aligned}$$

whenever $I_j = (\Delta_j, \psi_j, \Gamma_j)$, $j = 1, 2$, are two independent instances.

However, to construct an adequate set for $I_1[\vee\otimes]I_2$ or $I_1[\vee\uplus]I_2$, we need more than adequate sets for I_1 and I_2. A set $T \subseteq P(\Delta)$ is an **extended test set** for $I = (\Delta, \psi, \Gamma)$

if T is adequate for $(\Delta, \mathsf{true}, \Gamma)$ and $T \cap D$ is adequate for I. The following lemma and its corollary tie together extended and adequate sets. We leave the easy proof to the reader.

Lemma 5.3 *If T and T' are two extended test sets for I then so is $(T \cap D) \cup (T' \setminus D)$.*

Corollary 5.4 *If T is a minimal extended test set for I then $T \cap D$ is a minimal adequate set for I.*

The advantage of working with extended test sets is that extended test sets for $I_1[\vee\otimes]I_2$ and $I_1[\vee\uplus]I_2$ can be constructed from extended test sets for I_1 and I_2, and the construction preserves the property of being minimal. For $[\vee\otimes]$ we simply define $T_1[\vee\otimes]T_2 = T_1 \times T_2$. The definition of $T_1[\vee\uplus]T_2$ resembles that of $T_1[\vee\otimes]T_2$, but it is technically a bit more complicated; we omit the details here. With these definitions, Theorem 5.2 holds for extended test sets in place of adequate sets and for $\alpha = \vee\otimes$ or $\alpha = \vee\uplus$. Thus we can extend the approach in section 5.3 to large instances formed using $[\vee\otimes]$ and $[\vee\uplus]$. However, the operations $[\vee\otimes]$ and $[\vee\uplus]$ do not seem as useful in forming combined instances; typically one wishes to use the conjunction, not disjunction, of constraints.

6 Implementation issues

We have built a prototype implementation of a tool for generating adequate test sets. The tool reads an instance I of the test selection problem, and produces a minimal adequate set for I. The instances accepted by the tool are specified in the concrete language of Section 3.3; the criteria EACH and EXHAUSTIVE are also allowed, and are automatically converted to expressions that use only $\uplus$ and $\otimes$.

Internally, the tool works in six phases:

1. Parse the input and check its consistency (only declared parameters and values are used, no parameter is declared twice, etc.).

2. Eliminate EACH and EXHAUSTIVE.

3. Transform the criterion to the $\uplus\otimes$ form.

declaration

Alice : { A1, A2, A3, A4, A5 }
Bob : { B1, B2, B3, B4, B5 }
Cathy : { C1, C2, C3, C4, C5 }
Diana : { D1, D2, D3, D4, D5 }
Elaine : { E1, E2, E3, E4, E5 }

criterion

EXHAUSTIVE(**Alice**, **Bob**, **Cathy**, **Diana**, **Elaine**)

Figure 3: An instance to generate 3125 test points

4. Transform the constraint to the $\vee\wedge$ form.

5. Find a minimal adequate set, using the algorithm in Figure 2.

6. Print the test points.

The tool is implemented in C; the total size of the source files is about 1200 lines. The basic data structures are trees and forests, which are used to represent the parsed declarations, constraints and criteria, as well as the intermediate results for the transformations in phases 3 and 4.

We have tested the tool on RISC System/6000 Model 560, under the AIX operating system.[1] To measure the execution time on instances with large minimal adequate test sets, we have used the criterion EXHAUSTIVE. For the instance in Figure 3, the domain $D(\psi)$ has 3125 points, and the only adequate set is the whole domain. Although this is a very special form of a test selection criterion, the tool does not take any shortcuts; instances like this one are therefore suitable for performance measurements. The execution time of the tool for this input is slightly less than 30 seconds — that is, more than 100 test points per second. By using more sophisticated data structures we would be able to improve this number substantially; however, enhancing the functionality of the tool is more important than optimizing its running time.

[1]RISC System/6000 and AIX are trademarks of International Business Machines Corporation.

In particular, it would be worthwhile to extend the language with other data types (see the discussion of future work in Section 7.3). Other possible enhancements would be to add heuristics to the test selection algorithm, and to compute bounds on the size of the test set before the selection algorithm is invoked.

Although the tool always produces a *minimal* adequate set, it makes no attempt to come close to a *minimum* adequate set. A more sophisticated implementation would include **heuristics** to make the generated set smaller in "typical cases". A simple heuristic of this kind is to order the subcubes in the set S in Figure 2 so that smaller subcubes are processed before larger ones.

An approximate **bound** for the size of the produced test set would be useful as an early feedback to the user when the tool is used on a large instance. The user would appreciate some estimate of the size of the test set before the test selection algorithm itself is run. An upper bound can be easily computed as follows, even before phase 3 begins: In the criterion expression, replace each primitive criterion by the value 1, replace each $\uplus$ by the operator $+$, and each $\otimes$ by the operator $\times$. Then evaluate the resulting arithmetic expression; the result is an upper bound for the size of the minimal test set produced by the tool. An enhanced version of the tool would first display an initial (pessimistic) upper bound on the size of the test set, and then update the bound as the computation progresses. The designer could abandon execution if the bound seemed hopelessly large.

7 Concluding remarks

7.1 Related work

As is pointed out in the introduction, the representation of test selection criteria by sets of subsets of the input domain was considered, implicitly or explicitly, by a number of researchers. In *partition testing* [9], the input domain is partitioned into subsets, and one test point is then selected in each subset. This is an elaboration of the *condition table method* of Goodenough and Gerhart [5]. In this line of research, the emphasis has been on rules for constructing criteria from program texts and specifications. In contrast, the emphasis in the present paper is on a *language* for specifying criteria (i.e. sets of subdomains), and

on operations that allow test designers to combine criteria.

In his discussion of functional testing, Howden [8] stresses the need to identify input domains, and gives guidelines for systematic selection of test points for several types of input values that occur in scientific programs. Our basic philosophy is similar to Howden's; we develop this point of view further, by automating part of the selection process.

An important technical point is that we do not attempt to represent a criterion by a set of *disjoint* subsets. Note that our operation $\uplus$ would make little sense if we only considered sets of disjoint subsets. As is explained by Jeng and Weyuker [9], many naturally arising test selection criteria lead to non-disjoint sets of subdomains.

Gourlay [6] presents a precise framework for the discussion of issues in testing. In his terminology, our test selection criteria are a special form of the test methods for the set-choice construction testing system. Gourlay reinterprets previously published discussions about the suitability of various test selection criteria. In our approach, we do not attempt to decide a priori which criteria are sufficient — we leave that decision to the test designer. That is why we emphasize the importance of a *language* in which criteria are specified.

7.2 Comparison with TSL

Balcer, Hasling and Ostrand [2] describe a complete test language, called TSL, in which the test designer specifies a template for the test cases to be generated, *categories* (i.e. parameters and environment conditions), choices of values for the categories, and results of the test cases. A TSL specification is automatically translated to a set of individual test cases.

We now explain how TSL relates to the languages for test selection criteria that we propose in this paper. We will not describe TSL here; the reader is referred to the original paper [2] for a detailed description.

A TSL specification contains declarations of parameters, each with a set of values. (TSL makes a distinction between parameters and environment conditions, but for the purpose of this discussion both are considered to be parameters.) The specification also contains a set of Boolean conditions (the IF clauses in the RESULT sections), which are used to decide what combinations of parameter values are to be selected to form test cases. There are two

types of such conditions: unqualified ones, and those qualified by the directive SINGLE.

Let us first consider the following simplified form of the test selection criterion used by TSL: For an unqualified condition, all combinations of parameter values satisfying the condition should be selected. For a qualified condition, at least one combination of parameter values should be selected. We show how to specify this criterion in our language. Let $\varphi_1, \ldots, \varphi_m$ be the unqualified conditions, and let $\sigma_1, \ldots, \sigma_r$ be the conditions qualified as SINGLE. The test selection criterion is

$$\biguplus_{i=1}^{m} \left(\langle \varphi_i \rangle \otimes \mathsf{EXHAUSTIVE} \right) \uplus \biguplus_{j=1}^{r} \langle \sigma_j \rangle \tag{11}$$

If all φ_i and σ_j are conjunctions of conditions of the form

$$\begin{aligned} q &= c \\ q &\neq c \end{aligned}$$

where q is a parameter and c is a value of q, then the criterion (11) can be expressed in the concrete language from Section 3.3.

The TSL criterion as stated in [2] is actually more complicated than the one in the previous paragraph. An error-sensitizing rule is used to constrain the choice of a test point for $\langle \sigma_j \rangle$. The rule is described only informally in [2]; we now state one possible formalization, using our language. For each σ_j, $j = 1, \ldots, r$, let ω_j be the disjunction of all φ_i and σ_i in the same RESULT section, other than σ_j itself. The modified test selection criterion is

$$\biguplus_{i=1}^{m} \left(\langle \varphi_i \rangle \otimes \mathsf{EXHAUSTIVE} \right) \uplus \biguplus_{j=1}^{r} \langle \sigma_j \wedge \neg\, \omega_j \rangle$$

It is not our goal to discuss the merits of various versions of the error-sensitizing rule. We merely make the point that our language is a convenient notation for stating such rules precisely.

The language scheme proposed in this paper indicates the direction in which the TSL notation for test selection, and other similar notations, could be extended. The test designer

would benefit from the flexibility of the operations $\uplus$ and $\otimes$. For instance, in the example in Section 2, suppose that the test designer wants to fix **separator_1** $=$ ”/”, **separator_2** $=$ ”/” and **string_1_occurs** $=$ true, and test all values of **string_1** except ”” and all values of **string_2** at least once, but not necessarily all combinations of **string_1** and **string_2**. The criterion to express that requirement is

$$\begin{aligned} & \langle \textbf{separator\_1} = \text{"/"} \rangle \otimes \langle \textbf{separator\_2} = \text{"/"} \rangle \otimes \langle \textbf{string\_1\_occurs} = \mathsf{true} \rangle \\ \otimes \quad & \langle \textbf{string\_1} \neq \text{""} \rangle \otimes \ (\ \mathsf{EXHAUSTIVE}(\textbf{string\_1}) \uplus \mathsf{EXHAUSTIVE}(\textbf{string\_2})\ ) \end{aligned}$$

7.3 Future work

Here we mention several topics for further research which we have not addressed in the present paper. We group the topics into two categories: Improved algorithms for the concrete language, and extensions of the language and its use.

▷ **Algorithms for our concrete language**

In Section 5.3 we describe an algorithm for discovering a decomposition into independent instances. We assume that the constraint has the form $\bigwedge_k \psi_k$. To what extent can that assumption be relaxed?

Consider only the instances of the test selection problem that are built from instances of some small bounded size using the operations $[\wedge\otimes]$ and $[\wedge\uplus]$. Is there an efficient algorithm for finding minimum adequate sets for the instances in this special form?

Heuristics for finding “almost-minimum” adequate test sets for “common” test selection criteria should be investigated. In view of the results in Section 4.3, known heuristics for graph coloring would be a good starting point.

▷ **Extensions of the language**

The general language schema in Section 3.1 is a framework for further design of concrete languages based on other data types. After the enumerated data types treated in Section 3.3, the next most important type is **integers**. Some useful criteria for integers were mentioned in [11], but we have not studied in detail the algorithms needed to deal with those criteria.

Another important candidate for incorporation into the general schema is the type **words over a finite alphabet**, which would be useful for specifying criteria that have to do with control flow in a program or in a state machine.

The ideas in Section 5.3 lead naturally to *modular* descriptions of complex test suites. In a testing system supporting modularity, parameterized test cases along with test selection criteria could be created for various subsystems of a complex implementation under test, independently of each other (perhaps written by different test designers), and then combined by means of simple operators.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Co. 1974.

[2] M.J. Balcer, W.M. Hasling and T.J. Ostrand. Automatic generation of test scripts from formal test specifications. *Proc. Third ACM SIGSOFT Symp. Software Testing, Analysis and Verification* (Key West, Florida, December 13-15, 1989), 210-218.

[3] W.R. Elmendorf. Cause-effect graphs in functional testing. *IBM Poughkeepsie Laboratory, Technical Report TR-00.2487* (November 1973).

[4] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co., New York 1979.

[5] J.B. Goodenough and S.L. Gerhart. Toward a theory of test data selection. *IEEE Trans. Software Engineering SE-1* (1975), 156-173.

[6] J.S. Gourlay. A mathematical framework for the investigation of testing. *IEEE Trans. Software Engineering SE-9* (1983), 686-709.

[7] R. Hamlet. Theoretical comparison of testing methods. *Proc. Third ACM SIGSOFT Symp. Software Testing, Analysis and Verification* (Key West, Florida, December 13-15, 1989), 28-37.

[8] W.E. Howden. Functional program testing. *IEEE Trans. Software Engineering SE-6* (1980), 162-169.

[9] B. Jeng and E.J. Weyuker. Some observations on partition testing. *Proc. Third ACM SIGSOFT Symp. Software Testing, Analysis and Verification* (Key West, Florida, December 13-15, 1989), 38-47.

[10] G.J. Myers. *The Art of Software Testing.* John Wiley & Sons, Inc. 1979.

[11] J. Pachl. A notation for specifying test selection criteria. *Protocol Specification, Testing and Verification X* (L. Logrippo, R.L. Probert and H. Ural, editors; North-Holland 1990), 71-84.

[12] T.K. Philips. New algorithms to color graphs and find maximum cliques. *IBM Research Division, Research Report RC 16326* (November 1990).

[13] B. Toft. Colouring, stable sets and perfect graphs. *Handbook of Combinatorics*, to appear.